Information Sciences 565 (2021) 210–228

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Effective anytime algorithm for multiobjective combinatorial optimization problems

Miguel Ángel Domínguez-Ríos *, Francisco Chicano, Enrique Alba

ITIS Software, Universidad de Málaga, Spain

ARTICLE INFO

Article history:
Received 28 May 2020
Received in revised form 27 January 2021
Accepted 24 February 2021
Available online 5 March 2021

Keywords:
Multiobjective combinatorial optimization
Anytime algorithm
Well-spread non-dominated points

ABSTRACT

In multiobjective optimization, the result of an optimization algorithm is a set of efficient solutions from which the decision maker selects one. It is common that not all the efficient solutions can be computed in a short time and the search algorithm has to be stopped prematurely to analyze the solutions found so far. A set of efficient solutions that are well-spread in the objective space is preferred to provide the decision maker with a great variety of solutions. However, just a few exact algorithms in the literature exist with the ability to provide such a well-spread set of solutions at any moment: we call them *anytime* algorithms. We propose a new exact anytime algorithm for multiobjective combinatorial optimization combining three novel ideas to enhance the anytime behavior. We compare the proposed algorithm with those in the state-of-the-art for anytime multiobjective combinatorial optimization using a set of 480 instances from different well-known benchmarks and four different performance measures: the overall non-dominated vector generation ratio, the hypervolume, the general spread and the additive epsilon indicator. A comprehensive experimental study reveals that our proposal outperforms the previous algorithms in most of the instances.

1. Introduction

MultiObjective Optimization (MOO) is a field of research with many applications in different areas, such as biology, computer science, scheduling, and finances. Broadly speaking, in a multiobjective optimization problem some objectives are in conflict and they have to be optimized simultaneously. Without loss of generality, we consider minimization problems throughout this paper. In the case of a maximization problem, we use the property $max(f) = -min(-f)$. A MultiObjective Program (MOP) is an optimization problem characterized by multiple and conflicting objective functions that are to be optimized over a feasible set of decisions. If the constraints and the objective functions are linear, the MOP is a MultiObjective Linear Program (MOLP), and when the variables are integer, we name it MultiObjective Integer Program (MOIP). If some variables are constrained to be integer-valued and the rest are continuous, the problem is a MultiObjective Mixed Integer Program (MOMIP). MultiObjective Combinatorial Optimization (MOCO) is a special case of MOIP when the feasible set is finite. MOIP and MOCO are special cases of MultiObjective Discrete Optimization (MODO).

Finding the whole Pareto front in MOCO problems can require too much computational time when a faster solution is needed. For example, one may need to solve an assignment problem in hours and finding the whole Pareto front can take

* Corresponding author.
E-mail addresses: miguel.angel.dominguez.rios@uma.es (M.Ángel Domínguez-Ríos), chicano@lcc.uma.es (F. Chicano), eat@lcc.uma.es (E. Alba).

https://doi.org/10.1016/j.ins.2021.02.074
0020-0255/

days. In particular, this happens in instances with many non-dominated points, or when the model is hard in itself. From a practical point of view, the decision maker with whom we interact may be interested in a set of solutions as well-spread as possible in the objective space at any time during the search. The concept of *anytime* algorithms was introduced by Dean and Boddy [4] to characterize algorithms with the following two properties: *(i)* they can be terminated at any time and still return an answer, and *(ii)* the answers returned improve in some well-behaved manner as a function of time. In a more precise way, algorithms that show a better trade-off between solution quality and runtime are said to have a better anytime behavior [23].

In this paper we propose a new exact algorithm to solve MOCO problems, which is also valid for MODO problems with finite feasible sets. The fact that the algorithm is exact and all the solutions found are in the Pareto front allows less freedom when interpreting the algorithm and reduces the probability that two different implementations behave differently [31]. The algorithm is anytime in the sense that it is possible to interrupt its execution and take the non-dominated set. Moreover, it is guaranteed that this set is well-spread in the objective space. To check this spread and the quality of the solutions properly, four well-known metrics in the literature have been used: the overall non-dominated vector generation ratio [13], the hypervolume indicator [9,36], the general spread [13,37], and the additive epsilon indicator [21]. In the computational results section, all these metrics are calculated for each of the 480 instances considered in the work presented here.

The proposed algorithm is based on a general framework by Dächert and Klamroth [2]. Broadly speaking, it consists in analyzing a search region in the objective space and looking for new non-dominated points at each iteration. These regions can be considered as boxes in $\mathbb{R}^p$, where p is the number of objectives. The contribution of this paper is to present new designing criteria and an innovative way of splitting the objective space so that the solutions obtained are well-spread over the objective space. These are detailed in the following three novel proposals:

- A new strategy to select the appropriate search space region as the next box to explore, in order to guarantee that the new solutions found are spread over the objective space.
- A new way of partitioning the search space after finding a new non-dominated point. This partition reorders the selection of the future boxes to explore and also has an influence in the spread of solutions.
- A new quality function to measure the priority for the new regions to explore.

In order to help readers to reproduce the results of this paper, we uploaded the source code to GitHub.[1] This is a good practice that is becoming popular to face the reproducibility crisis in Computer Science [12].

The paper is organized as follows. In Section 2, we provide a literature review including the most relevant papers on MOO algorithms that could be used as anytime MOCO algorithms with good spread. In Section 3, we provide the background with some general definitions and the metrics we use in the computational results. In Section 4, a general formulation to solve MOCO problems is given. This formulation is the backbone of our contribution. In Section 5 our new anytime algorithm is presented. An exhaustive computational analysis is shown in Section 6. The last section is dedicated to the conclusions and future work.

2. Literature review

In this section we review different exact algorithms to solve multiobjective problems. We organize this review in four subsections. The first three subsections cover related works, which are presented (mostly) in chronological order in each subsection. The subsections focus respectively on the first algorithms, the ones based on recursion, and more recent algorithms. The fourth subsection summarizes the algorithms which can be considered as anytime with good spread and that are used to compare with our proposed algorithm.

2.1. First exact algorithms

In 2004, Sylva and Crema [32], working on the idea of Klein et al. [16], designed a new method to solve MultiObjective Integer Linear Programming (MOILP). They partitioned the solution space of dimension p by adding p binary variables and p constraints to exclude the previously-generated non-dominated points. About three years later, Sylva and Crema [33] formulated a variant of their previous work, finding well-spread subsets of non-dominated points in MOMIP. If the variables are integer, the whole Pareto front is obtained. To the best of the authors' knowledge, this was the first work to show dispersed solutions in the objective space, supported by computational results.

Masin and Bukchin [24] developed an algorithm for MOMIP, called DMA (Diversity Maximization Approach), which finds solutions by maximizing a proposed diversity measure and guarantees the generation of the complete set of efficient points.

In 2013, Lokman and Köksalan [22] proposed two improved algorithms based on the work of Sylva and Crema [32]. The algorithms iteratively generate non-dominated points and exclude the regions that are dominated by the previously-generated non-dominated points.

[1] Available at https://github.com/MiguelAngelDominguezRios/boxMO.

Ceyhan et al. [1], based on the work of Lokman and Köksalan [22], proposed some algorithms to find a small representative set of non-dominated points for MOMIP. The first of them, called SBA, is able to find the complete Pareto front for discrete problems if enough time is available. The main advantage of the SBA algorithm is that it produces very well-spread solutions over the objective space. As noted in this paper [1], the work of Masin and Bukchin [24] is similar to Sylva and Crema's method in [33], because they generate similar representative subsets of non-dominated points for MOMIPs. In fact, SBA outperformed Masin and Bukchin's algorithm.

2.2. Algorithms based on recursion

In 2003, Tenfelde-Podehl [34] proposed a recursive algorithm for MOCO problems with p criteria. Dhaenens et al. [6] proposed a new method to solve MOP, called K-PPM (Parallel Partitioning Method), which is based on the work of Lemesre et al. [19], called 2-PPM and valid only for the bi-objective case. This new approach is valid for any dimension $p \geqslant 2$, and could also be considered as an extension of the idea of Tenfelde-Podehl [34].

Özlen and Azizoğlu [25] developed a general approach to generate the Pareto front for MOIP problems, based in the ε-constraint method. They also used recursion to solve multiobjective problems with fewer objectives obtaining efficiency ranges for each objective. One drawback of the method in [25] is that it generates the same non-dominated point many times. The algorithm was enhanced in the work of Özlen et al. [26].

2.3. Modern algorithms

Laumanns et al. [18,17] proposed the first algorithm to calculate the Pareto front in MOP through the resolution of a number of single-objective problems that depend only on the number of solutions in the front. The work of Laumanns et al. [18] was based on the ε-constraint method. They obtained a bound $O\left(k^{p-1}\right)$ for the number of calls to the single-objective solver, where k is the number of non-dominated points and p the number of objectives. In [17], Laumanns et al. presented another algorithm for solving MOP using only lower bounds (for maximization problems) which leads to a fewer number of constraints. The algorithm by Kirlik and Sayın [14] was an improvement of the work of Laumanns et al. [18]. They changed the order in which the subproblems were solved. The search is managed over $(p-1)$-dimensional rectangles.

In the last decade, modern algorithms solving MOO problems have been developed. Apart from the aforementioned work by Ceyhan et al. [1], there are other important papers we should mention. Przybylski et al. [28] proposed an algorithm to compute the supported non-dominated extreme points for MOIP. This work was generalized by Przybylski et al. [29] to obtain the complete Pareto front for MOIP and is considered as a generalization of the two-phase method of Ulungu and Teghem [35]. Özpeynirci and Köksalan [27], among other authors, developed an algorithm to generate supported non-dominated points for MOMIP. Dächert and Klamroth [2] conceived an algorithm to solve MOO for $p = 3$. Their method is based on the work of Przybylski et al. [28,29]. They named *full m-split* the general framework for solving any MOO of dimension m. This algorithm, suitably adapted, is used in our work. Later, Klamroth et al. [15] improved *full m-split*, developing two new methods, called *RE* and *RA*, where they were concerned with eliminating or avoiding redundancies in the search zone explorations. Recently, Dächert et al. [3] have provided new theoretical insights into structural properties of the search region, assuming that a finite set of mutually non-dominated points has already been computed. The work provides better results than those in [15] for high dimensions ($p \geqslant 6$). In 2018, Holzmann and Smith [11] designed an algorithm to solve MODO problems using a weighted augmented Tchebycheff scalarization. This work is also based on the work of Dächert and Klamroth [2]. They improved the results of Kirlik and Sayın [14] and Özlen et al. [26].

2.4. Related algorithms that can be considered as anytime with good spread

As stated in the preceding review, only a few methods can be considered to have good spread over the objective space. This is in concordance with the work of Ceyhan et al. [1], which summarized in two the number of algorithms which can be used to calculate the complete front in MOCO problems [24,33]. If fact, with their SBA algorithm, they outperformed the results in [33]. The work of Holzmann and Smith [11] can also be considered as anytime, so we include it in our study.

We analyze which other algorithms in the literature can be slightly modified to obtain a well-spread set of non-dominated points at any time during the search. The algorithms which use the ε-constraint method do not seem to be good at spreading solutions over the objective space, because in the objective function, just one selected objective is optimized (Ch. 4 of [7]). Moreover, methods which use branch and bound as the general framework to obtain the Pareto front are widely surpassed by modern algorithms, which avoid repeating solutions [2]. In addition, those which calculate the nadir point in a first phase can be slow because computing the nadir point is an NP-hard problem in itself [10], so we do not consider these algorithms.

In conclusion, we have chosen two algorithms: SBA, from the work of Ceyhan et al. [1], and the method of Holzmann and Smith [11]. They are, to the best of the authors' knowledge, the algorithms that can potentially provide a well-spread set of non-dominated points at any time of execution.

3. Background

In this section, we first present some basic definitions to understand the paper, and then we define the four metrics used for the performance assessment in the computational results.

3.1. Definitions on multiobjective optimization

A MOCO problem can be defined as

$$\min f(x) = (f_1(x), \ldots, f_p(x)), \tag{1}$$

$$s.t. \quad x \in X \subset \mathbb{R}^n, \tag{2}$$

where x is the decision vector, $p \in \mathbb{N}$ is the number of objectives with $p \geqslant 2$, $f_i : X \to \mathbb{R}$ with $i = 1, \ldots, p$ are the objective functions, and $X \neq \varnothing$ denotes the feasible solution set, which is discrete and bounded.

The notion of optimality with several objective functions is considered in the sense of Pareto optimization. Given two vectors $x, y \in \mathbb{R}^p$, we say that x is *weakly dominated* by y if $y_i \leqslant x_i$ for all $i = 1, \ldots, p$ (denoted $y \leqq x$). If y weakly dominates x and $y_k < x_k$ for at least one $k \in \{1, \ldots, p\}$, then we say that y *dominates* x (denoted $y \leqslant x$) or x *is dominated* by y. If strict inequality holds for all $k \in \{1, \ldots, p\}$, then we say that y *strictly dominates* x (denoted $y < x$) or x is *strictly dominated* by y.

A feasible solution $x \in X$ is an *efficient solution* if there is no $y \in X$ such that $f(y)$ dominates $f(x)$. A solution $x \in X$ is called *weakly efficient* if there is no $y \in X$ such that $f(y)$ strictly dominates $f(x)$. The image of an efficient solution x is a *non-dominated point*, $z = f(x)$. The image of a *weakly efficient solution* x' is a *weakly non-dominated point*, $z' = f(x')$. The set of all efficient solutions in a MOCO problem is called *efficient set*, X_E, and its image is the Pareto front, $PF = f(X_E) \subseteq \mathbb{R}^p$. An efficient solution is *supported* if its image lies on the frontier of the convex hull of $PF \subseteq \mathbb{R}^p$. Equivalently, $x \in X$ is supported if it minimizes a weighted sum of the p objectives involving positive weights. Due to the fact that, in MOCO, many of the elements in X_E could lead to the same image, we are only interested in the set PF and one anti-image for each element of this set.

Let $z^1, z^2 \in \mathbb{R}^p$. We say that $z^1 <_{lex} z^2$ if there exists an index q, where $z^1_q < z^2_q$ and $q = min\{k | z^1_k \neq z^2_k\}$. The symbol $<_{lex}$ represents the strict *lexicographic order*.

Given a permutation σ and a vector function $f : X \to \mathbb{R}^p$, we denote by $f_\sigma = (f_{\sigma(1)}, f_{\sigma(2)}, \ldots, f_{\sigma(p)})$ the vector function where the objectives are reordered using σ. We say that $x \in X$ is a *lexicographic optimal solution* for permutation σ if there is no $y \in X$ with $f_\sigma(y) <_{lex} f_\sigma(x)$. There exists a maximum of $p!$ different lexicographic optimal solutions, one for each permutation.

Given a Pareto front, PF, the *ideal point* is $z^I = (z^I_1, z^I_2, \ldots, z^I_p)$, where $z^I_i = \min_{x \in X} f_i(x) = \min_{z \in PF} z_i, \ \forall i = 1, \ldots, p$, and the *nadir point* is $z^N = (z^N_1, z^N_2, \ldots, z^N_p)$, where $z^N_i = \max_{x \in X_E} f_i(x) = \max_{z \in PF} z_i, \ \forall i = 1, \ldots, p$. It is clear that z^I and z^N are a lower and an upper bound for the Pareto front. While the ideal point is found by solving p single objective optimization problems, the computation of the nadir point without knowing the whole Pareto front involves optimization over the efficient set, a very difficult problem in general [8].

Although it is common to use the term 'efficient solution' in the decision space and 'non-dominated point' in the objective space, sometimes we use the term 'solution' to refer to both spaces, when there is no possible confusion. We assume that $|PF| > 1$, otherwise the problem can be solved in one iteration with a single-objective optimization technique.

Given two vectors $l, u \in \mathbb{R}^p$ with $l < u$, we define the *box* $[l, u]$ as:

$$[l, u] = \{x \in \mathbb{R}^p | l_i \leqslant x < u_i, \forall i = 1, \ldots, p\}. \tag{3}$$

When the lower bound of a box is the ideal point in a Pareto front, we usually represent the box only with its upper bound, that is, $[z^I, u] = [u]$.

3.2. Performance assessment

The performance assessment of algorithms for multiobjective optimization is not a trivial issue. A good analysis comparing different multiobjective evolutionary algorithms is given in [38]. Recently, Li and Yao [20] published a survey of the quality evaluation for 100 different metrics. In [13], Jiang et al. categorize the MOO metrics into four groups: capacity metrics, convergence metrics, diversity metrics, and convergence-diversity metrics. Despite the aforementioned difficulty of designing anytime algorithms with good spread (effective anytime algorithms) because performance is often evaluated subjectively (see [23]), we have tried to take a representative group of metrics in which we can assume that the better the value of these metrics, the better the performance of the algorithm. Thus, we take one metric from each of the four categories defined by Jiang et al.

Capacity metrics [13] quantify the number or ratio of non-dominated solutions. We use here the Overall Non-dominated Vector Generation Ratio, which is defined as

$$\text{ONVGR}(N) = \frac{|N|}{|PF|}, \tag{4}$$

where $N \subseteq PF$ is the set of non-dominated solutions found by a run of a search algorithm and $|\cdot|$ is the cardinality of a set. Thus, ONVGR is a rational number in the interval $[0, 1]$ that represents the fraction of points of the Pareto front that were found by the search algorithm. For computing ONVGR, we need to know the total number of points in the Pareto front.

Convergence metrics measure the degree of proximity of the set of solutions to the complete front. The *additive epsilon indicator* gives the minimum additive factor by which the approximation set has to be translated in the objective space in order to weakly dominate the reference set [38,21]. We have scaled each objective to obtain a value in the range [0,1]. This additive epsilon indicator is defined as

$$\varepsilon_+(N) = \max_{x \in PF} \min_{y \in N} \max_{i=1,\ldots,p} \left(\frac{y_i - x_i}{r_i} \right), \tag{5}$$

where r_i is the range of objective i in N. This metric is also used in the computational experiments conducted by Ceyhan et al. [1] with the name of *coverage gap*.

Diversity metrics indicate the distribution and spread of solutions. From this group, we have selected the *general spread* metric, also cited in [13]. The original *spread* metric (Deb et al. [5]) calculates the distance between two consecutive solutions, which only works for 2-objective problems. An extension to any dimension, defined in [37], computes the distance from a point to its nearest neighbor. Let N be a set of non-dominated points and $\{e_i\}_{i=1}^m$ the extreme solutions,[2] which are the images of the lexicographic optimal solutions of the complete Pareto front, $\bar{d} = \frac{1}{|N|} \sum_{x \in N} d(x, N)$, where $d(x, N) = \min_{y \in N; y \neq x} d^*(x, y)$ and d^* denotes the Euclidean distance between two p-dimensional vectors. We define the *general spread* as

$$\Delta^*(N) = \frac{\sum_{i=1}^m d(e_i, N) + \sum_{x \in N} |d(x, N) - \bar{d}|}{\sum_{i=1}^m d(e_i, N) + |N|\bar{d}}. \tag{6}$$

Note that the metric Δ^* is a non-negative number. The lower its value, the more well-spread is N. Every lexicographic optimal solution not obtained in the execution produces a positive value in $\sum_{i=1}^m d(e_i, N)$ and a higher value for Δ^*.

From the group of *convergence-diversity metrics* we have selected the *hypervolume* indicator. Given a set of d non-dominated points, $N = \{z^1, z^2, \ldots, z^d\}$, the *hypervolume* HV is the measure of the region which is simultaneously dominated by N and bounded by a reference point $r \in \mathbb{R}^p$:

$$\mathrm{HV}(N, r) = volume\left(\bigcup_{j=1}^{d} [z^j, r] \right), \tag{7}$$

where *volume* is the Lebesgue measure in $\mathbb{R}^p$. This is the quality measure with the highest discriminatory power among the known unary quality measures [23,30,38]. There are many software packages that calculate the exact hypervolume, given a non-dominated set and a reference point [9,36]. The reference point can be taken as $r_i = \max_{j=1,\ldots,d} z_i^j \quad \forall\, i = 1, \ldots, p$. This is an acceptable choice when we have no information about the complete Pareto front. In the computational experiments presented in this paper, we have calculated the complete fronts for all the instances in order to have a fitted reference point. This means that we know the nadir point for every instance. As noted in the work of Li and Yao [20], there is still no consensus on how to choose a proper reference point for a given problem. To assign some importance to the extreme points, we add one unit to the nadir point, so the reference point is $r_i = z_i^N + 1 \quad \forall i = 1, \ldots, p$. Sometimes, it is useful to consider the percentage of the total hypervolume reached:

$$\mathrm{HVR}(N, r) = \frac{\mathrm{HV}(N, r)}{\mathrm{HV}(PF, r)}. \tag{8}$$

Given two subsets of non-dominated points, A and B, we write $A \succeq B$ if every $z^2 \in B$ is weakly dominated by at least one $z^1 \in A$. A metric is *Pareto compliant* if, for every two subsets A and B with $A \succeq B$ and $B \succeq A$, the metric value for A is not worse than the metric value for B. It is desirable for a metric to be Pareto compliant [39]. In other words, it must not contradict the order induced by the Pareto dominance relation. Of the four metrics considered, ONVGR, HV and ε_+ are Pareto compliant. This is equivalent to saying that, if we add a new element to the subset of solutions, then ONVGR and HV do not decrease and ε_+ does not increase. Nevertheless, Δ^* is not Pareto compliant, and we have to be more careful with its analysis.

4. General formulation for solving MOCO problems

In this section we describe the framework of Dächert and Klamroth [2] to solve MOCO problems. The framework can be found in Algorithm 1 Our proposed algorithm, described in Section 5, is based on this framework, but we add three modifications that let us get a better spread of solutions over the objective space. The method used by Holzmann and Smith [11] is also based on this framework. The idea of the method is to maintain a set of search zones, $\mathfrak{U}$, which are p-dimensional boxes. Every search zone is defined by its upper bound, because we consider that the ideal point is the lower bound.

[2] The original paper does not clarify what they mean by extreme points.

Algorithm 1: *General method for MOCO problems*

Input: P
Output: N
1: $N = \varnothing$
2: $\mathfrak{U} \leftarrow \{U\}$
3: **while** $(\mathfrak{U} \neq \varnothing)$ **do**
4: Select $B \in \mathfrak{U}$
5: **if** ($P(B)$ *is feasible*) **then**
6: $x^* \leftarrow$ Optimal solution of $P(B)$
7: $N = N \cup \{f(x^*)\}$
8: Update $\mathfrak{U}$
9: **else**
10: $\mathfrak{U} \leftarrow \mathfrak{U} - \{B\}$
11: **end if**
12: **end while**

At the beginning, the set N, containing the non-dominated points, is empty (Line 1), and the set of search zones contains the initial element U (Line 2), defined by an upper bound of the nadir point. The mathematical program used by the algorithm, P, is an input parameter. The algorithm then enters a loop while there is at least one zone to analyze. In each iteration of the loop it selects one zone (Line 4), solves the optimization problem, and, if a new non-dominated point is found, it is saved in N. Every time it finds a new non-dominated point, it updates the set $\mathfrak{U}$ accordingly (Line 8), so as to prevent repeated solutions in the future. Another goal of the updating procedure is to reduce the number of search regions at each iteration. More specifically, at least box B is extracted from $\mathfrak{U}$. The algorithm ends because, in MOCO problems, the number of non-dominated points is finite.

In what follows, we study the relevant elements of Algorithm 1 and analyze different options which may produce a well-spread set of solutions if we treat it as an anytime algorithm. We focus on three aspects of the general method that we think are important to consider.

4.1. Mathematical program

The selected mathematical program is one of the key aspects of the algorithm. Not all methods that solve multiobjective problems have the same performance. In fact, those which are based on the ε-constraint method (Ch. 4 of [7]) should not be good as effective anytime algorithms, unless we use a strategy for the selection of the next search zone to explore that guarantees a good dispersion of solutions in the objective space. The mathematical program used in the SBA method by Ceyhan et al. [1] is[3]:

$$\min_{x \in X} \left(\varepsilon \sum_{i=1}^{p} \omega_i f_i(x) - \alpha \right), \tag{9}$$

$$s.t. \quad f_i(x) \leqslant u_i - \alpha, \quad i = 1, \ldots, p, \tag{10}$$

$$\alpha \geqslant 0, \tag{11}$$

where α is a variable which represents the coverage gap, ε is a sufficiently small positive constant and $\omega_i > 0 \; \forall i$. If the program 9 to 11 has a solution, then it is efficient.

Another mathematical program, used in the work of Holzmann and Smith [11], is based on a weighted augmented Tchebycheff norm:

$$\min_{x \in X} \left(\max_{i=1,\ldots,p} \left(\omega_i \left| f_i(x) - z_i^I \right| \right) + \epsilon \cdot \sum_{i=1}^{p} \omega_i \left| f_i(x) - z_i^I \right| \right), \tag{12}$$

where z^I is the ideal point of the problem. The parameters ω and ϵ are chosen to make the program feasible. Depending on the objective value, we conclude whether the problem has a new solution inside the box or not. The reader is referred to the original paper [11] for more information.

We observed relevant performance differences between the two programs in a preliminary study. The weighted augmented Tchebycheff norm behaves better regarding the spread of the solutions in the objective space.

[3] We changed the program to adapt it to minimization problems.

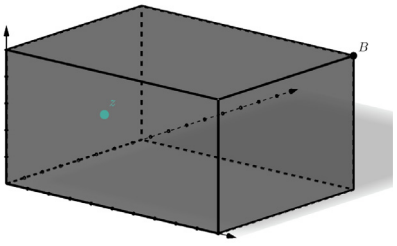
4.2. Selection of the next search zone to explore

Line 4 of Algorithm 1 proposes the extraction of the next search zone to analyze. The selection may be done randomly, but intuitively, if we split the objective space into different regions, and at each step we explore those which have a higher volume, we expect to obtain a final better spread of the solutions. Following this idea, we can define a numerical value for each search zone, called *priority*. One example of the *priority* function is the volume of the p-dimensional hyperrectangle with opposite vertices being the vector u (upper bound for the search zone) and z^I (ideal point). For instance, if $u = (10, 20, 30)$ and $z^I = (5, 12, 20)$, then the volume of the box is $(10 - 5) \cdot (20 - 12) \cdot (30 - 20) = 400$, and this could be its priority. A more sophisticated way of selecting the next box to explore is mentioned in Section 5.1.

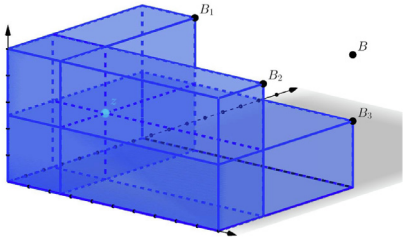
4.3. Updating the set of search zones

This may be the most important aspect to take into account in Algorithm 1 (Line 8) in terms of termination of the general algorithm and computational effort. One way to ensure that a non-dominated point is never computed again during the search consists in splitting the objective search zone and discarding at least that point. A good way to do this is found in the work of Dächert and Klamroth [2]. After a solution z is found, they divide the search zone into p new zones. The union of the new search zones discards the points dominated by z and the new search regions to explore contain at least one point less than the original search region. Given a box B with upper bound $u = (u_1, \ldots, u_p)$, let us suppose that we solve the mathematical problem and obtain a new non-dominated point $z = (z_1, \ldots, z_p)$. Then, we split the box B into $\{B_i\}_{i=1}^p$ with upper bounds $\{u^i\}_{i=1}^p$, being $u^i = (u_1, \ldots, u_{i-1}, z_i, u_{i+1}, \ldots, u_p) \forall i = 1, \ldots, p$. Each box B_i with upper bound u^i is said to have *direction* i. If the lower bound of all the boxes B and B_i is the ideal point, z^I, we call the split *full p-split*. A graphical example is shown in Figs. 1a and 1b for $p = 3$. Note that when a new non-dominated point is obtained, we must apply *full p-split* to all search zones that contain that new point, in order to assure the non-duplicity of the solutions. *Full p-split* is used in the work of Holzmann and Smith [11]. In Section 5.2 we describe a new way of splitting the objective space, called *p-partition*.

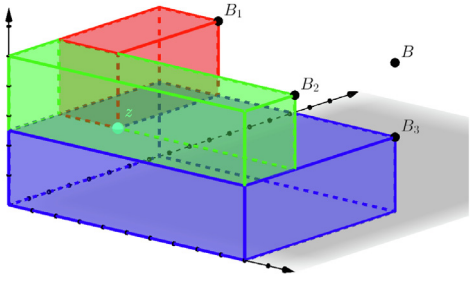
The splitting process often generates redundant zones. If we have two search zones u^1, u^2 with $u^1 \leqslant u^2$, all potential non-dominated points generated by u^1 could also be generated by u^2, which means that u^1 is redundant. Therefore, a filtering



(a) Box $\mathbf{B} = [(0, 0, 0), (20, 15, 10)]$ with non-dominated point $z = (5, 5, 5)$ inside.



(b) Resulting boxes after *full p-split*: $\mathbf{B_1} = [(5, 15, 10)]$, $\mathbf{B_2} = [(20, 5, 10)]$, and $\mathbf{B_3} = [(20, 15, 5)]$. All the lower bounds are $(0, 0, 0)$. Each box shares some space with another box.



(c) Resulting boxes after *p-partition*: $\mathbf{B_1^*} = [(0, 5, 5), (5, 15, 10)]$, $\mathbf{B_2^*} = [(0, 0, 5), (20, 5, 10)]$, and $\mathbf{B_3^*} = [(0, 0, 0), (20, 15, 5)]$. Each pair of boxes is disjoint.

Fig. 1. Splitting a box using *full p-split* and *p-partition*. Upper bound for the new boxes is the same in both cases, but lower bounds change.

process should be implemented after the split. Klamroth et al. [15] proposed two different algorithms for this purpose. The first is called *RE* (*redundancy elimination*) and consists in eliminating those redundant search zones at each iteration. The second algorithm is called *RA* (*redundancy avoidance*) and is based on structural properties of local upper bounds which yield necessary and sufficient conditions for a candidate local upper bound to be non-redundant. The filtering step is avoided in this case. In the computational experiments of the work by Klamroth et al. [15], they obtained better results in runtimes using *RE* when $p \in \{3, 4, 5\}$, and using *RA* when $p > 5$.

A more recent paper by Dächert et al. [3] describes a specific neighborhood structure among local upper bounds. With this structure, updates to the search region when a new non-dominated point is found can be more efficient compared to *RE* and *RA* approaches, as the number of points increases, but *RA* and even *RE* perform better for a small number of solutions. Since our proposal is designed as an anytime algorithm where the decision maker can stop the execution whenever desired, it is reasonable to think that the total number of solutions may not be high, unless a complete execution is derived. Thus, we have decided to use the *RE* approach in this paper.

5. Proposed anytime algorithm: *TPA*

In this section, we propose a new exact anytime algorithm based on the general framework described in Section 4 but with three novel contributions. This algorithm can solve any MOCO problem and it can also be used for MODO problems with finite feasible sets. We call it *TPA* because it uses a Tchebycheff mathematical program with Partition of the objective space using Alternating of directions. The pseudocode is shown in Algorithm 2.

Algorithm 2: *TPA*

Input: f and X // The MOCO problem
Output: XZ // A set of efficient solutions and their image
1: $\mathfrak{Q} = (\mathfrak{Q}_1, \ldots, \mathfrak{Q}_p)$ and $\mathfrak{Q}_i = \varnothing, \quad \forall i = 1, \ldots, p$
2: $XZ = \varnothing$
3: $k = 0$
4: Compute the ideal point and an estimation of the nadir point: z^I and $\bar{z}^N$
5: Set $\epsilon = 1/(2p(r-1))$ where $r = \max_{i=1}^{p}\{\bar{z}^N - z^I\}$
6: $\mathfrak{Q}_1.insert([z^I, \bar{z}^N])$
7: $P(u, \omega) \equiv \min_{x \in X}(\max_{i=1,\ldots,p}(\omega_i|f_i(x) - z_i^I|) + \epsilon \cdot \sum_{i=1}^{p}\omega_i|f_i(x) - z_i^I|)$
8: **while** $\exists i : |\mathfrak{Q}_i| > 0$ **and** *intime* **do**
9: $(B, k) \leftarrow$ Select_next_box($\mathfrak{Q}$, k)
10: Set $\omega = (\omega_1, \ldots, \omega_p)$ where $\omega_i = 1/\max\{1, B.u_i - z_i^I\}$
11: $(x^*, obj) \leftarrow$ Solve $P(B.u, \omega)$
12: **if** $obj < 1$ **and** $z^I < B.u$ **then**
13: $XZ \leftarrow XZ \cup \{(x^*, f(x^*))\}$
14: *Update* $(\mathfrak{Q}, f(x^*))$
15: **else**
16: $\mathfrak{Q}_k.remove(B)$
17: **end if**
18: **end while**

We now provide a high-level description of the algorithm and go in depth into the novel contributions in separate subsections. Every search zone is a box with the following fields: *lower bound* (l), *upper bound* (u), and *priority*. The vector $\mathfrak{Q}$ contains p priority queues (heaps) of boxes. The boxes in queue $\mathfrak{Q}_i$ have direction i and are sorted by non-increasing priority. The initial box has as lower and upper bounds the ideal point and an upper bound of the nadir point. Thus, it contains the whole Pareto front. We insert this box into $\mathfrak{Q}_1$ (Line 6), although any other queue $\mathfrak{Q}_i$ could be chosen. The ϵ value is constant during the execution and depends on the initial bounds (Line 5).

TPA then executes a loop while there is a box in any list of $\mathfrak{Q}$ and the time limit is not reached (Boolean *intime*). In the loop, it first selects the next box to analyze (Line 9) using Algorithm 3. Then, it solves the mathematical program in Eq. (12) (Line 7). As pointed in Section 4, this mathematical program provides better spread of the non-dominated points. The parameters are taken from the work of Holzmann and Smith [11]: $\omega_i = 1/\max\{1, u_i - z_i^I\}$, $\forall i$, and $\epsilon = 1/(2p(r-1))$, where u is the upper bound of the next box to analyze, $r = \max_{i=1,\ldots,p}(\bar{z}_i^N - z_i^I)$, z^I is the ideal point and $\bar{z}^N$ is an upper bound of the nadir point, computed as follows:

$$\bar{z}_i^N = \max_{x \in X}(f_i(x)) \qquad \forall i = 1, \ldots, p. \tag{13}$$

When $u \in \left[\bar{z}^N\right]$, the model is always feasible. Moreover, if $z^l < u$ the solution is inside the box $[u]$ if and only if the objective value is less than one unit (Theorem 5 of [11]). Thus, we can slightly modify Line 5 of Algorithm 1, changing the expression '*is feasible*' by '$obj < 1$ and $z^l < u$' (Line 12 of Algorithm 2). If it finds a new solution, x^*, it adds the pair $(x^*, f(x^*))$ to XZ (Line 13) and updates the boxes in the priority queues $\mathfrak{Q}$ (Line 14). Otherwise, it discards the box from the corresponding queue (Line 16).

The correctness and termination of *TPA* is proven in Theorem 1. This is an exact anytime algorithm and, thus, the complete Pareto front is found if the algorithm is given enough time. This guarantees the optimality and convergence of the algorithm. Observe that all the solutions found belong to the Pareto front. This contrasts with other heuristic and metaheuristic algorithms that can only compute approximate solutions without any guarantee of being efficient. Such algorithms require mechanisms to explore the search space that can be interpreted differently by different developers. Rostami et al. [31] have recently studied this issue and found that the resulting approximate solutions can influence the final results in such a way that the differences in quality metrics can even be statistically significant. The reliability and stability of *TPA* are supported by the ILP solver used. The memory usage and computation time are proportional to the number of boxes to maintain, and this can be exponential in the number of solutions found. Furthermore, the number of solutions in the Pareto front can be exponential in the number of decision variables. Thus, the memory required by the algorithm and the run time can be exponential in the number of variables if the whole Pareto front is computed. Regarding memory usage, in our experiments, using 2 GB of RAM was enough even when the complete Pareto front was computed.

In the next subsections we will detail the three novel contributions of the algorithm: a method to diversify the search exploring boxes with different directions, a new way to split the boxes forming a partition of the original one, and a new priority function for the boxes to explore.

5.1. Alternation of directions in the search

In this section we propose a new strategy for the selection of the next box to explore, trying to diversify the regions of the objective space that are explored. This is the first novel contribution of this paper. The priority queue $\mathfrak{Q}_i$ contains boxes with *direction i*. Boxes in different directions are displaced in the objective space along the different dimensions. We propose to select a box from a different priory queue $\mathfrak{Q}_i$ in each iteration of *TPA*. Among the ones with the same direction *TPA* chooses the box with higher *priority*. The pseudo-code of the Select_next_box procedure (Line 9 in Algorithm 2) is in Algorithm 3.

Algorithm 3: Select_next_box

Input: $\{\mathfrak{Q}_i\}$, k // List of boxes and integer counter
Output: (B, k)
1: $B = \varnothing$
2: **if** $\exists i : |\mathfrak{Q}_i| > 0$ **then**
3: $k = (k \bmod p) + 1$
4: **while** $|\mathfrak{Q}_k| = 0$ **do**
5: $k = (k \bmod p) + 1$
6: **end while**
7: $B \leftarrow \text{argmax}_{B \in \mathfrak{Q}_k}(B.priority)$
8: **end if**

5.2. New updating procedure of the search zone

This section presents a new way of splitting the box, called *p-partition*. This is the second contribution of the paper. The main difference between *p-partition* and *full p-split* is that the new boxes created by *p-partition* form a partition of the original box (they are pairwise disjoint).

Definition 1. Let $B = [l, u]$ be a box and $z \in \mathbb{R}^p$ a point. For every $i = 1, \ldots, p$, we define $B_i(z) = \{x \in B | x_i < z_i \text{ and } x_j \geqslant z_j \forall j > i\}$. We also define $B_0(z) = \{x \in B | x_j \geqslant z_j \ \forall j\}$.

It is easy to check that $B_i(z)$ are boxes, which can be written as follows:

$$B_i(z) = \left[\left(l_1, \ldots, l_i, \hat{z}_{i+1}, \ldots, \hat{z}_p\right), \left(u_1, \ldots, u_{i-1}, \hat{z}_i, u_{i+1}, \ldots, u_p\right)\right], \tag{14}$$

where $\hat{z}_i = \max\{z_i, l_i\}\ \forall i = 1, \ldots, p$. If $B_i(z) \neq \varnothing$, then it has direction i.

Proposition 1. *Given a box B, then* $\{B_i(z)\}_{i=0}^{p}$ *is a partition of B.*

Proof. We need to prove that $B_i(z) \cap B_j(z) = \varnothing$ for all $i \neq j$ and $\bigcup_{i=0}^{p} B_i(z) = B$. Suppose $x \in B_i(z) \cap B_j(z)$ for indexes i and j, where $i < j$. As $x \in B_i(z)$, then $x_j \geqslant z_j$. As $x \in B_j(z)$, then $x_j < z_j$. This is a contradiction, so the sets are always disjoint and the first part is proved.

Let $x \in \bigcup_{i=0}^{p} B_i(z)$. By definition, $x \in B$, so $\bigcup_{i=0}^{p} B_i(z) \subset B$. Now let $x \in B$. If $x_i \geqslant z_i\ \forall i$, then $x \in B_0(z)$. Otherwise, let j be the higher index which verifies $x_j < z_j$. Then, $x \in B_j(z) \subset \bigcup_{i=0}^{p} B_i(z)$, so $\bigcup_{i=0}^{p} B_i(z) = B$ and the second part is proved. □

Definition 2. Let $B = [l, u]$ be a box, and $z < B.u$. We define the *p-partition* of the box B according to z as $\{B_i(z)\}_{i=1}^{p}$.

When obtaining a non-dominated point z, the exploration of some other boxes in the future might produce the same solution. In order to prevent this, we split those boxes and remove the space weakly dominated by z, which is $B_0(z)$. That is why *p-partition* does not include $B_0(z)$.

Fig. 1 shows a graphical example of *p-partition* compared to *full p-split*. Note that the upper bounds in the resulting boxes are the same, but lower bounds are different.

It is very important to eliminate redundant boxes because, otherwise, the computational execution time grows exponentially. For *p-partition*, we apply a variant of the redundancy elimination (*RE*) used in *full p-split*. In the *RE* method, when a box A is contained in a box B, we eliminate A from the priority queue of boxes to explore. For a deep study of how this method works, see [15]. Nevertheless, it is not possible to apply this idea to *p-partition* because the lower bounds of the new boxes differ. We join boxes to save this obstacle.

Definition 3. Let $A = \left[l^a, u^a\right]$ and $B = \left[l^b, u^b\right]$ be two boxes with $u^a \leqslant u^b$. We define the *join box* of the two boxes as a new box $C = \left[l^c, u^c\right]$, where $l_i^c = \min\left\{l_i^a, l_i^b\right\}$ and $u_i^c = u_i^b\ \forall i = 1, \ldots, p$.

Once we introduce join boxes, we cannot guarantee that all the boxes are pairwise disjoint, but we can confirm that the new box does not exclude non-dominated points. We prove this in the next proposition. The *priority* of the new box C could be greater than the value of the previous boxes A and B. This means that box C will have higher preference to be analyzed in subsequent iterations.

Proposition 2. *With the conditions of Definition 3, it holds that* $A \cup B \subset C$*, which means that analyzing box C substitutes the analysis of the other two boxes.*

Proof. Let $x \in A \cup B$. If $x \in A$, then $l_i^c \leqslant l_i^a \leqslant x < u_i^a \leqslant u_i^b = u_i^c\ \forall i$, so $x \in C$. If $x \in B$, then $l_i^c \leqslant l_i^b \leqslant x < u_i^b = u_i^c\ \forall i$, so $x \in C$. □

We are now ready to present in Algorithm 4 the updating procedure of Line 14 in Algorithm 2. It is divided into two parts. In the first part (Lines 1–9), the boxes in $\mathfrak{Q}$ are split using *p-partition*. Some of the new boxes may be empty. The non-empty boxes are inserted into the corresponding $\mathfrak{Q}_i$ according to their direction (Line 6). The priority function used for the boxes is detailed in the next subsection. The second part (Lines 10–17) filters the redundant boxes using *RE* but taking into account that if a domination relation between two upper bounds is detected, the boxes are joined and the new box replaces the older ones (Lines 12–15). We finish this subsection proving the correctness and termination of *TPA*.

Proposition 3. *Algorithm TPA never finds the same non-dominated point twice.*

Proof. Let B be the box being explored in *TPA* and z the non-dominated point found solving $P(B.u, \omega)$ in Line 11 of Algorithm 2. The definition of the mathematical program P ensures that $z < B.u$, and the use of *p-partition* with each box B' such that $z < B'.u$ discards $B'_0(z)$, which is the only box that can contain z. The upper bound of a join box is always the upper bound of a previous box. Thus, after the updating procedure, the remaining boxes do not contain z and z cannot be found again. □

Algorithm 4: Update

Input: $(\mathfrak{Q}, z)$
Output: $(\mathfrak{Q}, z)$
for all $B \in \bigcup_{k=1}^{p} \mathfrak{Q}_k$ with $z < B.u$ **do**
  for $i = 1$ **to** p **do**
    $B_i(z) = \{x \in B | x_i < z_i \text{ and } x_j \geqslant z_j \forall j > i\}$
    if $(B_i(z) \neq \varnothing)$ **then**
      $B_i(z).priority \leftarrow reduced\_scaled(B, i, z)$
      $\mathfrak{Q}_i.insert(B_i(z))$
    end if
  end for
end for
for $i = 1$ **to** p **do**
  for all $A, B \in \mathfrak{Q}_i$ with $A.u \leqslant B.u$ **do**
    $C \leftarrow join(A, B)$
    $\mathfrak{Q}_i.remove(A)$
    $\mathfrak{Q}_i.remove(B)$
    $\mathfrak{Q}_i.insert(C)$
  end for
end for

Theorem 1. *TPA terminates after finding all the non-dominated points of the MOCO problem if the time limit is not set.*

Proof. The number of non-dominated points is finite in MOCO problems. By Proposition 3, no points are found twice. Thus, *TPA* terminates. Proposition 2 ensures that all *PF* is found if enough time is given. □

5.3. A new priority function

As the third contribution, we define a new priority function for the boxes. We want to give more priority to boxes having a larger region where non-dominated points can potentially exist. Given a non-empty box $B = [l, u]$, we define the *scaled* function as

$$scaled(B) = \prod_{i=1}^{p} \left(\frac{u_i - l_i}{\overline{z}_i^N - z_i^I} \right). \tag{15}$$

Note that $scaled(B) > 0$. In the case of an empty box, we define $scaled(B) = 0$. We assume, without loss of generality, that $\overline{z}_i^N > z_i^I \; \forall i$, otherwise the objective i always takes the same value and can be discarded.

In some boxes, there is a region that cannot contain any non-dominated point. In this case, the *scaled* function overestimates the priority of the box. We define a new priority function to correct this overestimation. First, we need some results.

Proposition 4. Let $B = [l, u]$ be a box and $z \in B$, then the box $[l, z] \subseteq B_p(z)$, where p is the dimension of the objective space.

Proof. From the definition of box, $z < u$. Let $x \in [l, z]$. Then, $l_i \leqslant x_i < z_i < u_i \; \forall i = 1, \ldots, p$. More specifically, $x_p < z_p$. This means that $x \in B_p(z)$, so $[l, z] \subseteq B_p(z)$. □

If $z = l$, we have $[l, z] = \varnothing$, and $B_i(z) = \varnothing$ for all $i = 1, \ldots, p$. The box $[l, z]$, if it is not empty, contains points dominating z. Thus, if z is non-dominated, the box $[l, z]$ does not contain any non-dominated point, otherwise z is dominated and we have a contradiction.

Definition 4. Let $B = [l, u]$ be a box, $z < B.u$ a non-dominated point and $B_i(z)$ for $i = 1, \ldots, p$ the box with direction i after *p-partition*. Then, for every $i = 1, \ldots, p$, we define the *reduced_scaled* function as

$$reduced\_scaled(B, i, z) = \begin{cases} scaled(B) & \text{if} \quad i \neq p \\ scaled(B) - scaled([l, z]) & \text{if} \quad i = p. \end{cases} \tag{16}$$

Note that some of the boxes after *p-partition* may be empty.

Proposition 5. *The reduced_scaled value is always non-negative.*

Proof. For $i = 1, \ldots, p-1$, it is obvious that *reduced_scaled* is non-negative (see Eq. (15)). Let us now assume that $i = p$. If $[l, z] = \varnothing$, then $scaled([l, z]) = 0$ and $reduced\_scaled(B_p(z), p, z) = scaled(B_p(z)) \geqslant 0$. If $[l, z] \neq \varnothing$, then $z \in B$ and Proposition 4 ensures that $[l, z] \subseteq B_p(z)$, so $scaled([l, z]) \leqslant scaled(B_p(z))$, and *reduced_scaled* is non-negative. □

We use the *reduced_scaled* function as the *priority* value of a box.

6. Computational results

In this section, we present a deep experimental study divided in four subsections. The first subsection describes the benchmark of instances we have used. The instances are grouped into categories. Each category has a fixed number of variables. The categories are also grouped into classes, where each class corresponds to a different MOCO problem. The authors have considered the choice of this benchmark appropriate because different MOCO problems could have different performances. The second subsection describes the parameters for the three algorithms used in the comparison. The third subsection shows the results of the computational experiments. For each instance and algorithm, the results are displayed at some time points. We provide a deep analysis of the results in the last subsection, including a statistical validation.

6.1. Benchmark

The benchmark we use is divided into three classes of well-known multiobjective problems. The first is the multiobjective knapsack problem (*KP*), the second is the multiobjective assignment problem (*AP*), and the third class corresponds to multiobjective integer linear programming problems (*ILP*). All the details of the formulation of the three classes can be found in [14], at URL http://home.ku.edu.tr/moolibrary/, and in the supplementary material. For each class, different problem categories are generated based on problem size. There are 10 instances in each category. In total, there are 480 instances. As stated in Section 3, the metrics we use to compare the algorithms are ONVGR, HV, Δ^*, and ε_+. To calculate these quality indicators, we need to calculate the complete Pareto front. The details of these experiments are shown in the supplementary material of the paper.

6.2. Algorithms and parameters

We downloaded and used the source code[4] provided by Ceyhan et al. [1]. This algorithm will be called *ceyhan* henceforth. The algorithms of Holzmann and Smith [11] (called *holzmann*) and *TPA*[5] were programmed using C++. Since *holzmann* and *TPA*, are taken from the same framework, we use in them the same filtering process of the solutions, the *RE* method, in order to make a fair comparison and observe the differences of performance with our new contributions.

The computer used for the experiments is a multicore machine with four Intel Xeon CPUs (E5-2670 v3) at 3.1 GHz, a total of 48 cores, 64 GB of memory and Ubuntu 16.04 LTS. For each run we used only 1 core and 2 GB of RAM.

The three algorithms use CPLEX 12.6.2 as the ILP solver. We set the CPLEX parameters, CPXPARAMEPGAP=CPXPARAMEPAGAP=CPXPARAMEPINT=0 and CPXPARAMPARALLELMODE=CPXPARAMTHREADS=1. Parameter CPXPARAMEPINT indicates the integrality tolerance for the solution variables. Its default value is 10^{-5}. Parameter CPXPARAMEPGAP sets a relative tolerance on the gap between the best integer objective and the objective of the best node remaining in the tree used in the *branch and cut*. The default value is 10^{-4}. CPXPARAMEPAGAP sets an absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. The default value is 10^{-6}. CPXPARAMPARALLELMODE is set to deterministic mode, which means that multiple runs with the same model and the same parameter settings on the same platform will reproduce the same solution path and results. Finally, CPXPARAMTHREADS sets the default number of parallel threads that will be invoked by any CPLEX parallel optimizer. In this case, it is set to 1.

Although the three algorithms are deterministic and the time limit is fixed, the number of solutions found can differ in different executions because CPLEX manages some internal parameters, such as the remaining available memory of the machine, which can influence the tree exploration to obtain the next solution. To soften these differences, we did 30 executions for each instance and algorithm, and then reported the average values.

6.3. Summary of the results

We calculated the four metrics in each algorithm (*ceyhan*, *holzmann*, *TPA*), for a fixed time in the set $\{10, 60, 300, 900\}$, measured in seconds. We have chosen this set of cut-times to know the performance of each algorithm at the very beginning of the run (just 10 s), and also after a longer (but still short) time (900 s), plus some other intermediate times.

[4] Commit 6b3b7e7bb9 on 17 Aug 19 at https://github.com/gokhanceyhan/MOIP_Solvers.
[5] Source code available at https://github.com/MiguelAngelDominguezRios/boxMO.

Table 1
Number of times that each algorithm obtains the best average value without exclusivity (*best*) and with exclusivity (*excl*), for ONVGR and HV, for different execution times, and for each class in the benchmark. Total sum is displayed in the last four rows.

		ONVGR						HV					
		ceyhan		*holzmann*		*TPA*		*ceyhan*		*holzmann*		*TPA*	
		best	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*
AP	t = 10 s	11	0	34	21	79	66	11	0	45	32	68	55
	t = 60 s	20	0	31	8	92	69	20	0	32	9	91	68
	t = 300 s	29	0	66	33	67	34	29	0	33	0	100	67
	t = 900 s	33	0	52	7	93	48	33	0	45	0	100	55
ILP	t = 10 s	44	8	116	55	157	96	43	7	119	58	155	94
	t = 60 s	51	3	134	48	169	83	50	2	120	33	185	98
	t = 300 s	61	1	146	35	184	73	60	0	129	18	202	91
	t = 900 s	76	0	158	19	201	62	76	0	146	7	213	74
KP	t = 10 s	41	0	79	22	138	81	41	0	95	38	122	65
	t = 60 s	56	0	97	4	156	63	56	0	93	1	159	67
	t = 300 s	71	0	128	0	160	32	71	0	127	0	159	33
	t = 900 s	80	0	149	0	160	11	80	0	148	3	156	12
TOTAL	t = 10 s	96	8	229	98	374	243	95	7	259	128	345	214
	t = 60 s	127	3	262	60	417	215	126	2	245	43	435	233
	t = 300 s	161	1	340	68	411	139	160	0	289	18	461	191
	t = 900 s	189	0	359	26	454	121	189	0	339	10	469	141

Table 2
Number of times that each algorithm obtains the best average value without exclusivity (*best*) and with exclusivity (*excl*), for Δ^* and ε_+, for different execution times, and for each class in the benchmark. Total sum is displayed in the last four rows.

		Δ^*						ε_+					
		ceyhan		*holzmann*		*TPA*		*ceyhan*		*holzmann*		*TPA*	
		best	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*	*best*	*excl*
AP	t = 10 s	75	64	29	17	19	7	11	0	27	14	86	73
	t = 60 s	59	39	40	20	41	21	20	0	25	2	98	75
	t = 300 s	44	15	43	13	72	42	29	0	34	1	99	66
	t = 900 s	54	21	36	3	76	43	33	0	45	0	100	55
ILP	t = 10 s	143	107	75	27	86	38	44	8	133	68	144	79
	t = 60 s	163	115	81	21	84	24	51	3	137	44	173	80
	t = 300 s	168	108	87	11	101	25	61	1	144	32	187	75
	t = 900 s	179	103	100	6	111	17	76	0	160	21	199	60
KP	t = 10 s	121	80	63	14	66	17	42	1	94	37	122	65
	t = 60 s	123	67	70	3	90	23	56	0	97	5	155	63
	t = 300 s	139	69	86	2	89	5	71	0	127	0	159	33
	t = 900 s	145	66	93	2	92	1	80	0	148	3	156	12
TOTAL	t = 10 s	339	251	167	58	171	62	97	9	254	119	352	217
	t = 60 s	345	221	191	44	215	68	127	3	259	51	426	218
	t = 300 s	351	192	216	26	262	72	161	1	305	33	445	174
	t = 900 s	378	190	229	11	279	61	189	0	353	24	455	127

Tables 1 and 2 present the results for each algorithm and metric. There are two columns per combination. The first one contains the number of instances in which the algorithm has the best average value (sometimes shared with another algorithm). The second column contains the number of instances in which the algorithm is the only one obtaining the best average value. The results are displayed for the three classes of the benchmark (*AP*, *ILP* and *KP*) at four cut-points each, and the total sum is shown in the four last rows of the table.

Analyzing the ONVGR metric, we can see that *holzmann* and *TPA* provide the higher number of solutions. In fact, *TPA* is better in all the classes in the benchmark. If we observe the row with the total sum of the three classes, the new proposed algorithm is clearly the best. Sometimes the number of solutions is not the most important issue. In fact, the hypervolume is considered one of the best metrics to measure the spread over the objective space. Looking at the results for HV, we see that in all the classes the total hypervolume reached by *TPA* is also maximum. It is the clear winner, with a large difference with respect to the second-best algorithm, *holzmann*. Note that *ceyhan* has the best hypervolume in a few cases. For instance, in the *ILP* class, at 10 s, *ceyhan* has the best value 43 times, but only in 7 cases it is the unique winner. In the other 36 instances, its HV is similar to that of other algorithms. The great differences in HV may be explained because, overall, *ceyhan* finds much fewer solutions than the others and, consequently, a worse total hypervolume. *Ceyhan* is good at finding a few solutions quickly, with a very good general spread, as we can see in the results for Δ^* in Table 2, where it is the clear winner. For the last metric, ε_+, Table 2 shows again that *TPA* has the best results in all cases.

As a summary of the results shown in Tables 1 and 2, we see that *holzmann* and *TPA* provide better performance than *ceyhan*. This means that the structure of the algorithm has an influence on the final results. While *ceyhan* is designed to find a few well-spread solutions quickly, the dispersion of the solutions in the objective space is not so good when the search progresses. *Ceyhan* makes more than one call to the ILP solver at each iteration because it uses a subset of vectors which depends on the number of solutions found so far. We noticed in the experiments that these multiple calls to the solver per iteration seem to degrade performance as the search progresses. The other two methods (*holzmann* and *TPA*), based on the framework described in Algorithm 1, seem to be more efficient for longer run execution times. There is an exception in the results for Δ^*, where *ceyhan* is the clear winner. Comparing *holzmann* and *TPA* methods, the latter obtains better results. Although these two algorithms are based on the same framework and use the same mathematical program, *TPA* changes the way in which the next boxes to explore are selected: by alternating directions, using *p-partition*, and using a different priority value assigned to each box. The results demonstrate that the three new contributions described in this paper have a positive effect on the anytime performance.

6.4. Detailed analysis

This subsection is divided into three third-level sections. In the first third-level section, we rank the algorithms according to their performance on the instances. Clarifying graphs are shown at ten regular cut-points from 90 s to 900 s. In the second third-level section, we show the evolution of the metrics over time for three selected instances. In this way, we observe the anytime performance of the algorithms for each metric. The last third-level section provides the results of non-parametric statistical tests to check if the differences are statistically significant or not. In the supplementary material attached to this paper, we group the instances into categories, and show tables with the results for the four metrics at four time points.

6.4.1. Ranking the algorithms in each class of instances

To better measure the performance of the algorithms, we computed a rank for each of them in each instance. We did this for ONVGR, HVR, Δ^*, and ε_+, at different stopping times from 90 s to 900 s, with a step of 90 s. For example, fixing a time limit of 360 s, we compared the average ONVGR value of the 30 runs of the algorithms in the first instance and assigned a rank

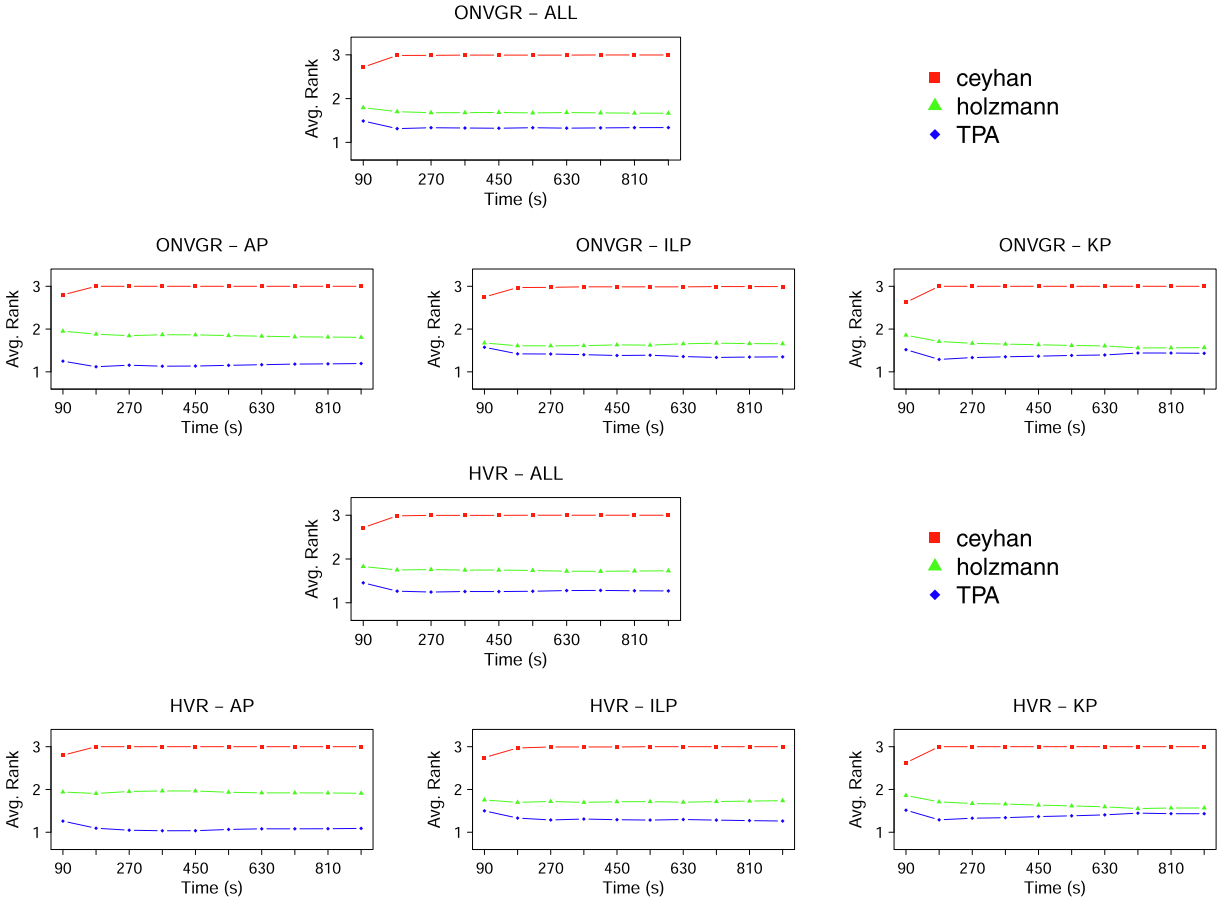


Fig. 2. Average rank values for the algorithms at 10 cut-points, using the metrics ONVGR and HVR. The lower the rank, the better the algorithm.

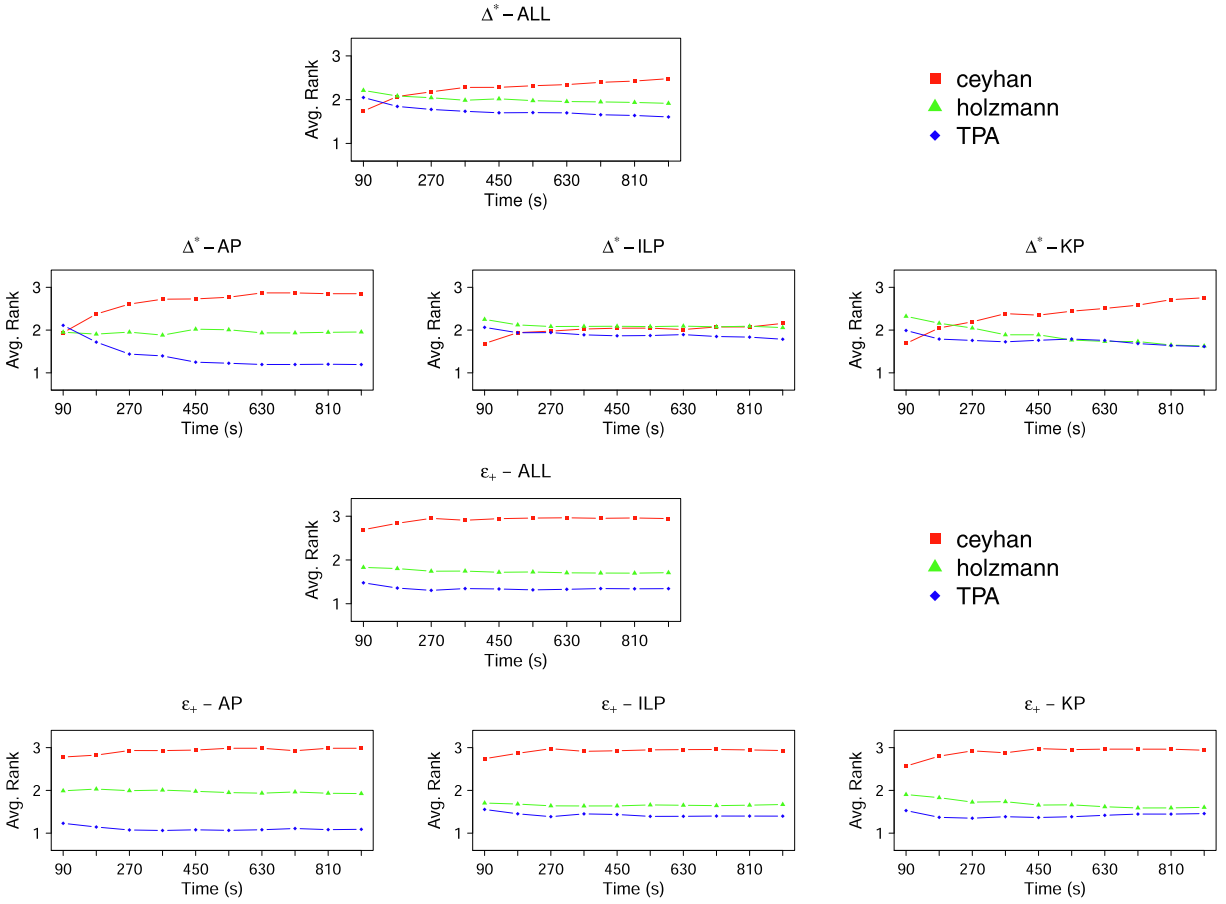


Fig. 3. Average rank values for the algorithms at 10 cut-points, using the metrics Δ^* and ε_+. The lower the rank, the better the algorithm.

from 1 to 3, being 1 the algorithm with the best value for the metric. We did the same in the rest of the cases. In the case of ties, we proceeded by taking the average rankings for the ties (e.g., 1.5 for a tie between the first two algorithms).

There are some time points in which an algorithm may have finished its execution and others not. In those cases, a rank value of 1 is assigned to the algorithm that finished and the following rankings are distributed among the rest. When all the algorithms have finished at a given time in all the runs, the instance is discarded. The results of Figs. 2 and 3 indicate that the best average rank values correspond to *TPA* in most of the cases with important differences compared to the other metrics in some cases.

6.4.2. Search progress

In this section we want to show the progress of the search on some selected instances, to further illustrate how the algorithms evolve and behave at any time. For each class of the benchmark, the instance which requires the longest time to be solved is selected.

Fig. 4 shows the behavior of ONVGR and HVR for the three selected instances: *KP_p-3_n-100_ins-4*, which belongs to the multiobjective knapsack problem group, in this case, with 100 variables; *AP_p-3_n-50_ins-4*, from the assignment problem group, which has 50 agents; and *ILP_p-4_n-80_m-40_ins-9*, from the *ILP* group, which has dimension 4, 80 variables, and 40 constraints. Fig. 5 displays the behavior for the other metrics. In the *x*-axis, we show the execution time in seconds. The *y*-axis shows the corresponding quality metric. In this case, the time points are taken every 5 s, up to 900 s. For these particular instances, *TPA* provides the higher number of solutions at any time, HVR is quite similar for *TPA* and *holzmann*, Δ^* is better for *ceyhan* in most of cut-points, and ε_+ is similar for the three methods. We observe a high increase in HVR during the first minutes and then a stabilization of the value at the end. We see that the evolution of Δ^* in Fig. 5 is not monotonic. This is a consequence of Δ^* not being Pareto compliant.

6.4.3. Statistical validation

In order to check if the observed differences are statistically significant or not, we applied the non-parametric Friedman test to compare the three algorithms. We applied the test for each metric and cut-point in the set {90,180,. . .,900}. In the

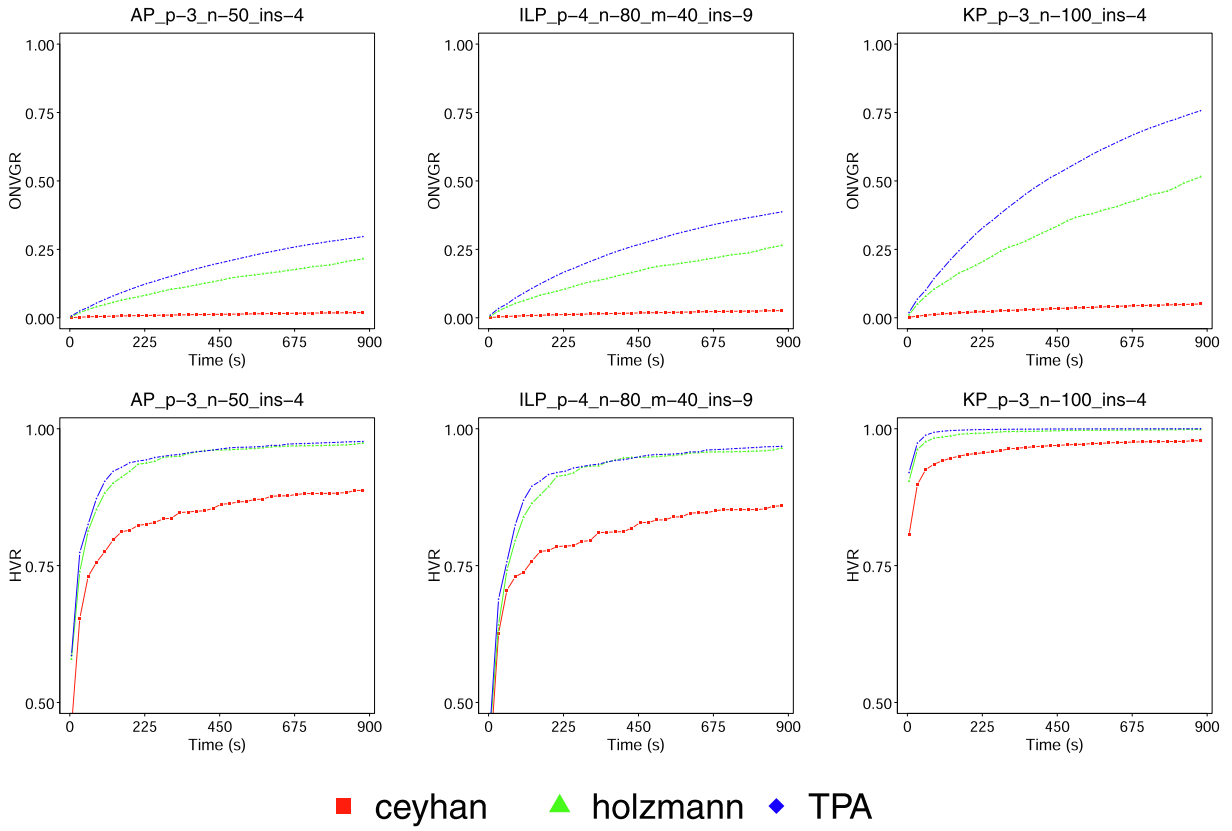


Fig. 4. ONVGR and HVR as a function of time in 3 selected instances. The higher the value, the better the algorithm.

cases in which the p-value is below the significance level $\alpha = 0.01$, there is a strong evidence that the performances of the algorithms are different. To pinpoint the differences, we did a post hoc analysis using the Nemenyi multiple comparison test. This test compares every pair of algorithms looking for significant differences in a paired test with the 480 instances, as well as for each class. We chose only the cases in which one method is significantly different than the other two, and then we applied the Wilcoxon signed rank test to confirm that the corresponding algorithm is the best. All these tests are available in the R-package *PMCMRplus*.

A summary with the results of the tests is shown in Table 3. We used the initial of the algorithm (C, H or T) in the cases where an algorithm has the best performance with a significance level $\alpha = 0.01$ in the three tests (Friedman, Nemenyi and Wilcoxon). We highlight in boldface the cases in which the p-value is below 0.001.

The first conclusion is that *TPA* is almost always the best algorithm. The second conclusion is that *Holzmann* is never the best for all the metrics. Another conclusion from the tests is that the behavior of the algorithms is problem-dependent. We observe that, for *ILP* and *KP* problems, *ceyhan* is significantly better with respect to Δ^*. In *AP* instances, *ceyhan* is the best in the first 180 s but the differences with *TPA* are not statistically significant. From 270 s to 900 s, *ceyhan* is surpassed by *TPA*. Grouping all the instances, the tests are statistically significant at a significance level of $\alpha = 0.001$ at any cut-point for three of the four metrics, in favor of *TPA*. The algorithm proposed in this paper shows a statistically significant difference with *holzmann* for these quality indicators.

7. Conclusions and future work

We have designed a new anytime algorithm that allows the Pareto front to be calculated in multiobjective combinatorial optimization problems. The front is well-spread at any time and has good values for the metrics ONVGR, HV, Δ^*, and ε_+. We have first carried out an exhaustive study of the literature, and have identified two state-of-the-art methods which produce a well-spread set of non-dominated points at any time and are not outperformed by other methods. They were proposed by Ceyhan et al. [1] and Holzmann and Smith [11].

The algorithm we propose, called *TPA*, is based on an existing framework to solve MOCO problems [2], which has been adapted to design an effective anytime algorithm. The new contributions are: the establishment of a new strategy to select the appropriate search space region as the next box to explore, a new way of partitioning the search space after finding a new

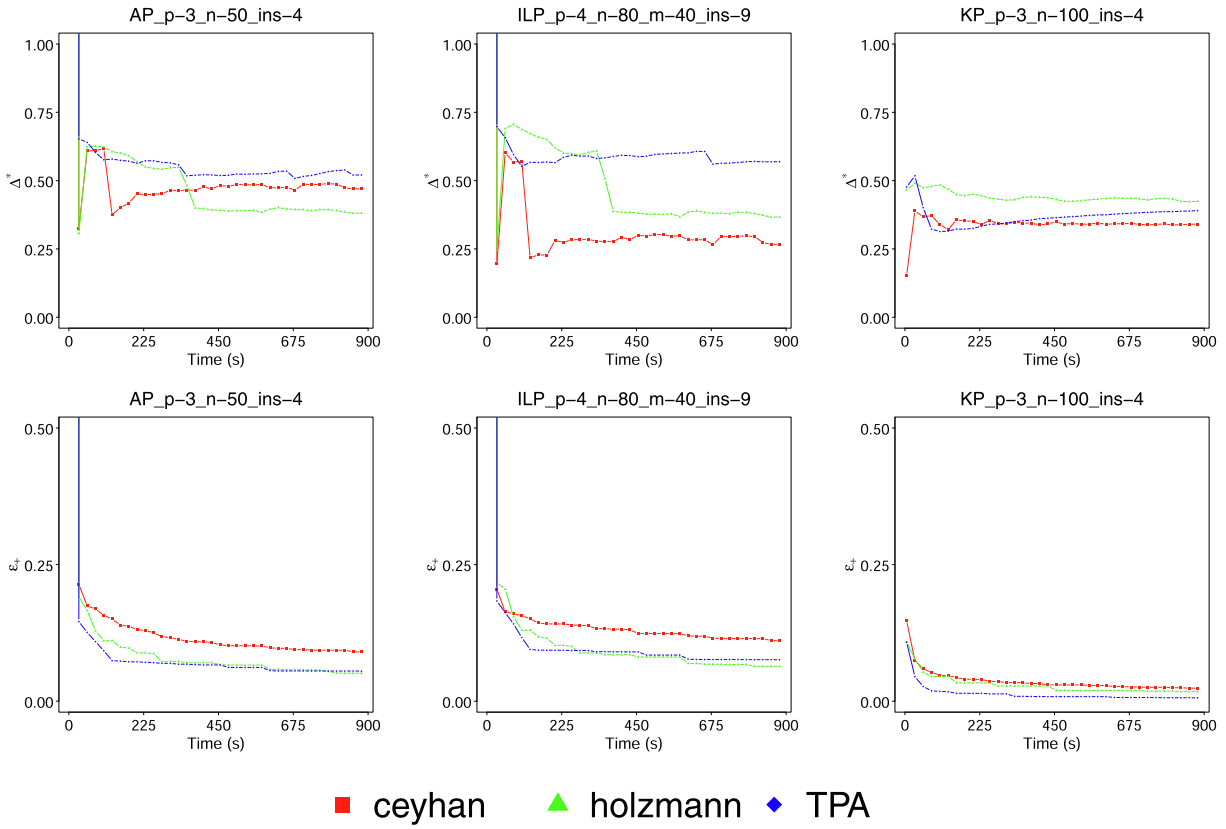


Fig. 5. Δ^* and ε_+ as a function of time in 3 selected instances. The lower the value, the better the algorithm.

Table 3
Statistical tests for each class and metric at different time points. The initial of the best algorithm is shown when the differences with the others are statistically significant at level $\alpha = 0.01$. If the differences are also significant at level $\alpha = 0.001$, the initial of the algorithm is marked in bold.

Hypothesis tests		90s	180s	270s	360s	450s	540s	630s	720s	810s	900s
AP	ONVGR	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	HVR	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*		T	T	**T**	**T**	**T**	**T**	**T**	**T**	
	ε_+	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
ILP	ONVGR							T	T	T	T
	HVR		**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*	**C**	**C**	**C**	**C**	**C**	**C**	**C**	**C**	**C**	C
	ε_+						T	T		T	T
KP	ONVGR	T	**T**	T	T	T	T				
	HVR	T	**T**	T	**T**	T	T				
	Δ^*		**C**	**C**	C	**C**	**C**	**C**	C		
	ε_+	**T**	**T**	**T**	T	T	T				
TOTAL	ONVGR	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	HVR	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**
	Δ^*	**C**									
	ε_+	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**	**T**

non-dominated point, and the definition of a new quality function to set the priority for the new regions to explore. These three new contributions have a positive influence in the spread of the solutions.

We compared *TPA* with *ceyhan* and *holzmann* in several ways. A deep performance analysis was done using 480 instances and executing the algorithms 30 times in each of them. We compared the algorithms using four different metrics well-known in the MOO literature. Finally, we have statistically checked if the observed differences are statistically significant using Friedman test, Nemenyi multiple comparison test, and Wilcoxon signed rank test. We can statistically confirm (for this benchmark of instances) that *TPA* is better than the state-of-the-art methods in almost all the cases, improving the number of

solutions, hypervolume and additive epsilon indicator for all the time cut-points used, which indicates a better anytime behavior. In the case of the general spread (which is not Pareto compliant), *ceyhan* is better in the first 90 s, and there is no conclusion for the rest of cut-points when considering all the instances.

Future work includes the extension of the experimental study to other benchmarks, to analyze in which kind of problems our proposed algorithm works best. We can also apply *TPA* to multi-objective industrial problems, where a few efficient solutions covering the objective space are required by decision makers in a short time. The ideas introduced in this paper can also be applied to heuristic algorithms, in the field of evolutionary multi-objective optimization, for example, to improve the performance of multi-objective meta-heuristics. We can also combine heuristics with exact methods to create new hybrids enjoying the advantages of both fields.

CRediT authorship contribution statement

Miguel Ángel Domínguez-Ríos: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing - original draft, Visualization. **Francisco Chicano:** Conceptualization, Writing - review & editing, Supervision, Resources, Funding acquisition. **Enrique Alba:** Writing - review & editing, Supervision, Resources, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been partially funded by the Spanish Ministry of Economy and Competitiveness (MINECO) and the European Regional Development Fund (FEDER) under contract TIN2017-88213-R (6city project), the European Research Council under contract H2020-ICT-2019-3 (TAILOR project), the University of Málaga, Consejería de Economía y Conocimiento de la Junta de Andalucía and FEDER under contract UMA18-FEDERJA-003 (PRECOG project), the Ministry of Science, Innovation and Universities and FEDER under contract RTC-2017-6714-5, and the University of Málaga under contract PPIT. UMA.B1.2017/07 (EXHAURO Project).

Appendix A. Supplementary data

Supplementary data associated with this article can be found, in the online version, at https://doi.org/10.1016/j.ins.2021.02.074.

References

[1] G. Ceyhan, M. Köksalan, B. Lokman, Finding a representative nondominated set for multi-objective mixed integer programs, Eur. J. Oper. Res. 272 (1) (2019) 61–77.
[2] K. Dächert, K. Klamroth, A linear bound on the number of scalarizations needed to solve discrete tricriteria optimization problems, J. Global Optim. 61 (4) (2015) 643–676.
[3] K. Dächert, K. Klamroth, R. Lacour, D. Vanderpooten, Efficient computation of the search region in multi-objective optimization, Eur. J. Oper. Res. 260 (3) (2017) 841–855.
[4] T. Dean, M. Boddy, Solving time-dependent planning problems, in: Proceedings of the 7th National Conference on Artificial Intelligence, 1988, pp. 49–54.
[5] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, in: International Conference on Parallel Problem Solving from Nature, Springer, 2000, pp. 849–858.
[6] C. Dhaenens, J. Lemesre, E. Talbi, K-PPM: a new exact method to solve multi-objective combinatorial optimization problems, Eur. J. Oper. Res. 200 (1) (2010) 45–53.
[7] M. Ehrgott, Multicriteria Optimization, vol. 491, Springer Science & Business Media, 2005.
[8] M. Ehrgott, D. Tenfelde-Podehl, Computation of ideal and nadir values and implications for their use in MCDM methods, Eur. J. Oper. Res. 151 (1) (2003) 119–139.
[9] C.M. Fonseca, L. Paquete, M. López-Ibáñez, An improved dimension-sweep algorithm for the hypervolume indicator, in: Proceedings of the IEEE Congress on Evolutionary Computation, IEEE, 2006, pp. 1157–1163.
[10] M.R. Gary, D.S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, 1979..
[11] T. Holzmann, J.C. Smith, Solving discrete multi-objective optimization problems using modified augmented weighted Tchebychev scalarizations, Eur. J. Oper. Res. 271 (2) (2018) 436–449.
[12] M. Hutson, Artificial intelligence faces reproducibility crisis, Science 359 (6377) (2018) 725–726.
[13] S. Jiang, Y. Ong, J. Zhang, L. Feng, Consistencies and contradictions of performance metrics in multiobjective optimization, IEEE Trans. Cybern. 44 (12) (2014) 2391–2404.
[14] G. Kirlik, S. Sayín, A new algorithm for generating all nondominated solutions of multiobjective discrete optimization problems, Eur. J. Oper. Res. 232 (3) (2014) 479–488.
[15] K. Klamroth, R. Lacour, D. Vanderpooten, On the representation of the search region in multi-objective optimization, Eur. J. Oper. Res. 245 (3) (2015) 767–778.
[16] D. Klein, E. Hannan, An algorithm for the multiple objective integer linear programming problem, Eur. J. Oper. Res. 9 (4) (1982) 378–385.
[17] M. Laumanns, L. Thiele, E. Zitzler, An adaptive scheme to generate the Pareto front based on the epsilon-constraint method, in: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2005..

[18] M. Laumanns, L. Thiele, E. Zitzler, An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method, Eur. J. Oper. Res. 169 (3) (2006) 932–942.
[19] J. Lemesre, C. Dhaenens, E. Talbi, Parallel partitioning method (PPM): a new exact method to solve bi-objective problems, Comput. Oper. Res. 34 (8) (2007) 2450–2462.
[20] M. Li, X. Yao, Quality evaluation of solution sets in multiobjective optimisation: a survey, ACM Comput. Surv. 52 (2) (2019) 1–38.
[21] A. Liefooghe, B. Derbel, A correlation analysis of set quality indicator values in multiobjective optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 581–588.
[22] B. Lokman, M. Köksalan, Finding all nondominated points of multi-objective integer programs, J. Global Optim. 57 (2) (2013) 347–365.
[23] M. López-Ibáñez, T. Stützle, Automatically improving the anytime behaviour of optimisation algorithms, Eur. J. Oper. Res. 235 (3) (2014) 569–582.
[24] M. Masin, Y. Bukchin, Diversity maximization approach for multiobjective optimization, Oper. Res. 56 (2) (2008) 411–424.
[25] M. Özlen, M. Azizoğlu, Multi-objective integer programming: a general approach for generating all non-dominated solutions, Eur. J. Oper. Res. 199 (1) (2009) 25–35.
[26] M. Özlen, B.A. Burton, C.A.G. MacRae, Multi-objective integer programming: an improved recursive algorithm, J. Optim. Theory Appl. 160 (2) (2014) 470–482.
[27] Ö. Özpeynirci, M. Köksalan, An exact algorithm for finding extreme supported nondominated points of multiobjective mixed integer programs, Manage. Sci. 56 (12) (2010) 2302–2315.
[28] A. Przybylski, X. Gandibleux, M. Ehrgott, A recursive algorithm for finding all nondominated extreme points in the outcome set of a multiobjective integer programme, INFORMS J. Comput. 22 (3) (2010) 371–386.
[29] A. Przybylski, X. Gandibleux, M. Ehrgott, A two phase method for multi-objective integer programming and its application to the assignment problem with three objectives, Discrete Optim. 7 (3) (2010) 149–165.
[30] S. Rostami, F. Neri, A fast hypervolume driven selection mechanism for many-objective optimisation problems, Swarm Evol. Comput. 34 (2017) 50–67.
[31] S. Rostami, F. Neri, K. Gyaurski, On algorithmic descriptions and software implementations for multi-objective optimisation: a comparative study, SN Comput. Sci. 1 (5) (2020) 1–23.
[32] J. Sylva, A. Crema, A method for finding the set of non-dominated vectors for multiple objective integer linear programs, Eur. J. Oper. Res. 158 (1) (2004) 46–55.
[33] J. Sylva, A. Crema, A method for finding well-dispersed subsets of non-dominated vectors for multiple objective mixed integer linear programs, Eur. J. Oper. Res. 180 (3) (2007) 1011–1027.
[34] D. Tenfelde-Podehl, A recursive algorithm for multiobjective combinatorial optimization problems with q criteria, Universität Graz/Technische Universität Graz. SFB F003-Optimierung und Kontrolle, 2003..
[35] E.L. Ulungu, J. Teghem, The two phases method: an efficient procedure to solve bi-objective combinatorial optimization problems, Found. Comput. Decis. Sci. 20 (2) (1995) 149–165.
[36] L. While, L. Bradstreet, L. Barone, A fast way of calculating exact hypervolumes, IEEE Trans. Evol. Comput. 16 (1) (2012) 86–95.
[37] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, E. Tsang, Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion, in: 2006 IEEE International Conference on Evolutionary Computation, IEEE, 2006, pp. 892–899.
[38] E. Zitzler, L. Thiele, M. Laumanns, C.M. Fonseca, V.G. Da Fonseca, Performance assessment of multiobjective optimizers: an analysis and review, IEEE Trans. Evol. Comput. 7 (2) (2003) 117–132.
[39] E. Zitzler, D. Brockhoff, L. Thiele, The hypervolume indicator revisited: on the design of Pareto-compliant indicators via weighted integration, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2007, pp. 862–876.